



Distributed Interactive Computing Environment

Quick Start

For those wishing to dive in and give **DICE** a try before reading any explanation, try the following :

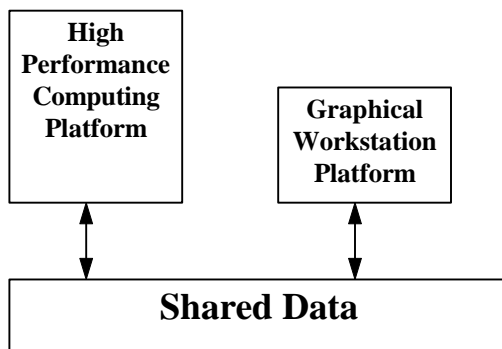
- **mkdir** /usr/local/Dice
- **cd** /usr/local/Dice
- **tar** xvf dice.tar
- **setenv** DICE_ROOT /usr/local/dice
- **source** \$DICE_ROOT/.dicerc
- **dice**

For those interested in a little background, read on.

Introduction

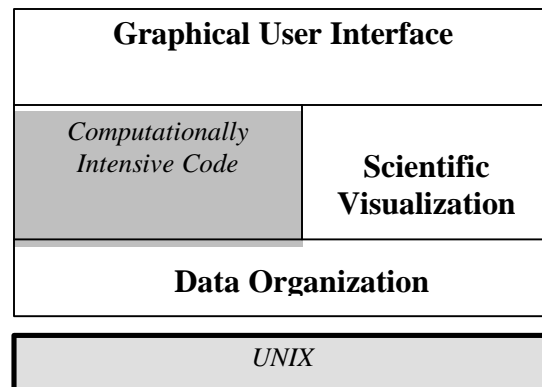
The *Distributed Interactive Computing Environment* (**DICE**) is a flexible toolkit for developing high performance computing applications. **DICE** also includes some canned applications in the areas of Computational Structural Mechanics, Computational Fluid Dynamics and Computational Chemistry.

The purpose of **DICE** is to take advantage of distributed computing resources; using each to it's full advantage, for the purpose it was designed.



Typically, a computationally intensive code will be run on a large high performance platform. Graphical user interface and scientific visualization, however, perform better on resources with resources with robust graphical facilities.

DICE allows the computationally intensive code to run on a large parallel platform while allowing the user to interactively interrogate the runtime data from a graphics workstation. This is accomplished by allowing the data to be logically shared but physically distributed across one or more platforms.



DICE consists of three major sections : Data Organization, Scientific Visualization, and Graphical User Interface tools. Combined with existing or new computational codes, **DICE** is used to develop a complete application environment.

Organization

The foundation for *DICE* is a system known as *Network Distributed Global Memory* (NDGM). NDGM presents one or more processes with a contiguous, unstructured memory buffer that may span the physical memory of several machines.

On top of NDGM is the *DICE Data Directory* (DDD). This uses NCSA's **Hierarchical Data Format** to implement data structures on NDGM. The *DICE Data Directory* provides support for structured and unstructured data as well as runtime information and grouping of datasets.

Graphical User Interface (GUI) development is accomplished via the Tcl/Tk scripting language. Support for *NDGM* and *DDD* has been added to Tcl/Tk in addition to an object oriented drag and drop capability and support for an OpenGL viewer.

Another important *DICE* component is scientific visualization. A number of data filters (isosurface generators and data slicers) and a generalized viewing system called *DICE Visualizer* (DV) has been developed to support scientific visualization within the environment.

DICE is a modular system. This allows separate portions of a *DICE* application to execute on the most appropriate resource. For example, the computationally intensive portion may execute on a massively parallel processor (MPP) while the visualization and GUI execute on a user's workstation.

Three computational areas have currently been targeted for inclusion into *DICE*. A Computational Fluid Dynamics code, *Zns* (Zonal Navier-Stokes), has been incorporated into an application as well as a Computational Structural Dynamics code (CTH), and a Computational Chemistry Code. All applications allow real time interactive visualization of a running code in a distributed environment.

Distributed Shared Memory

NDGM is a client-server system that consists of multiple server processes and an Application Programmers Interface (API) for clients. Each server maintains a section of a virtual contiguous buffer and fields requests for data transfer and program synchronization. Clients use the API to transfer data in and out of the virtual buffer and to coordinate their activity.

Calls to the API result in lower level messages being sent to the appropriate NDGM server which keeps track of its piece of the total virtual buffer. The API translates the global memory address into a local address which the server then transfers from its local memory.

Client programs use the API to access the virtual NDGM buffer as contiguous bytes. No structure is placed upon the NDGM buffer; the application can impose any structure on this buffer that is convenient. In addition, NDGM is designed to implement a system of applications in contrast to a single monolithic parallel application. The API includes facilities to get and put contiguous memory areas, get and put vectors of data, acquire and release semaphores, and to initialize and check into multiple barriers.

For synchronization purposes, the API provides barriers and semaphores. Checking into a barrier will result in the process blocking until the barrier value reaches zero. Requesting a semaphore will block until it is released by the client currently owning the requested semaphore.

All requests for data transfer and synchronization are handled by the NDGM server process. This is a stand-alone program that waits for new connections from clients and services their requests.

Each server maintains a local memory buffer which maps into the virtual buffer address space. This local buffer can be in one of three locations: local address space (obtained via malloc), system shared memory, or a local file. If system shared memory is used, a client executing on the same physical machine as the server accesses the shared memory instead of making requests to a server. This access is transparent to the NDGM client application and results in faster data transfers. Using a file as the server's local storage, allows NDGM servers to restart with their local memory already initialized.

Clients and servers run on top of a layered message passing interface. Similar in concept to well known message passing interfaces like PVM or MPI, this layer provides a level of abstraction, freeing the upper layers from the details of reading and writing data. The NDGM message passing layer has fewer facilities than either PVM or MPI but is designed to pass NDGM data efficiently with minimal copying. This layer provides calls to establish connections, send messages, probe for incoming messages, read messages, and close connections.

The actual interprocess data transfer is accomplished by the "drivers". Current drivers include: TCP/IP sockets, PVM, and Fifos. An Intel NX driver is currently available for use on the Intel Paragon. Each driver has functions to open as a client, open as a server, read, write, and probe for incoming messages. When possible, each driver also implements a "select" function in order to monitor several open connections. A single NDGM system can mix nodes utilizing different drivers.

DICE Data Directory

NCSA has developed a data format and access routines for scientific datasets. The routines allow users to create, read, write, and query large datasets. Support exists for structured, and unstructured datasets. In addition, there are "Vgroups" which allow for the grouping of datasets.

NCSA's Hierarchical Data Format (HDF) has been modified to allow support for NDGM. The I/O facility known as the "Low Level H Interface" has been modified to read and write data from NDGM as well as from disk files. To accomplish this, a *Domain* is prepended to the filename before it is opened. For example, to open a disk file named data.hdf, the string "file:data.hdf" is passed to the HDF routines. To put the same file in NDGM, the string "NDGM:data.hdf" is used.

The *DICE Data Directory* (DDD) is a layer of routines on top of HDF that further organizes the data. DDD supports structured and unstructured data as well as hierarchical groupings of these data into directories. In addition, DDD supports a *Control* data which is intended for cooperative control of running codes.

In addition to the base data types, DDD supports "data references". Data References are subsets of base data types that allow a level of abstraction that conceptually simplifies the dataset. For example, suppose we have a block CFD values around a projectile. We have 5 values for each of a 100x100x100 dataset. So the entire dataset is 100x100x100x5. We could define a data reference named "Pressure on surface" that is 100x1x100x1 that represents only the pressure on the surface of the projectile. This data can now be referenced as a 2D dataset; the lower levels of DDD handle the mapping of requested access to actual target data.

Graphical User Interface Tools

A major portion of *DICE* is dedicated to providing the user with a flexible but powerful Graphical User Interface (GUI). A scripting language is used to access the various GUI tools. An unstructured scripting language allows complex interfaces to be rapidly prototyped and promotes re-use of modules due to its lack of strict typing. In contrast to structured, strictly typed languages like JAVA, scripting languages can provide complex yet independent and reusable modules for non time critical applications like a GUI. JAVA and 'C' with Motif execute faster than scripting languages and may be better for fixed, well

defined user interfaces. But for rapid prototyping and independent, malleable, reusable modules, a scripting language is the right tool for the job.

Tcl/Tk is a popular scripting language available on a variety of platforms. Tcl (Tool Command Language) provides basic language constructs like assignment, flow control, and I/O. Tk provides a windowing system interface which allows the creation and management of widgets like buttons, labels and scales.

Extensions to the Tcl/Tk language which are included in *DICE* are :

- *Tix* - Convenience widgets like FileBoxes and Tree widgets
- *BLT* - Drag and Drop facility, 2D graphs and extended I/O

Tcl/Tk allows for 'C' routines to be compiled into the interpreter for time critical functions. Support for NDGM, HDF, DDD and the *DICE* Visualizer have been included in this fashion. This allows for time critical functions be efficiently serviced while maintaining the advantages of using a scripting language.

Tcl/Tk in conjunction with Tix and BLT allow for the rapid development of complex GUIs. For example, in addition to simple widgets like buttons and labels, there is a 2D plotting widget to support line graphs within an interface. This widget allows the user to easily re-graph subsets of the data or to query specific data points. This plotting widget is combined with a spreadsheet widget to give the user a complete data query interface.

A major portion of the *DICE* GUI toolkit is dedicated to the "Drag and Drop" facility. This allows object oriented interfaces to be designed and for extensive re-use of developed code. Objects, represented by icons, can be selected, dragged and dropped onto targets in other windows of the interface. These objects, known as "sources", generally represent a DDD item. Sources contain all information necessary to retrieve the underlying DDD item, but not the actual data. Targets generally represent functions that operate on DDD items. For example, there is an information target that will display name, dimensions, and other pertinent information about any source that is dropped.

The Drag and Drop interface allows generic tools to be developed that are not specific to any one application. Tools for 2D plots and isosurfaces have been developed that will accept sources that represents a DDD item. Any new GUI only need to reference these tools from a menu bar to incorporate the functionality. In addition, this makes the operation of an isosurface generator in a flow code consistent with the isosurface generator in a structural dynamics code.

Scientific Visualization

Scientific Visualization within *DICE* is achieved via the *DICE Visualizer* (DV) and a variety of data filters. DV is a viewing system consisting of a world through which a scientist can steer, to investigate visual objects that represent the data. The data filters are routines by which an application's datasets can be transformed into a meaningful visual objects. These scientific visualization components within *DICE* have been specifically designed to enable the visualization of large datasets at interactive rates.

DV is designed to quickly visualize large volumes of data. This data is stored in the DDD in the form of unstructured datasets and then read into DV where it is processed into display ready items (like polygons), displayed and manipulated.

The internal structure of DV is composed of 4 distinct modules: the object management module, the graphics module, the navigator module and the input module. Conceptually, these modules have very different functions; however to ensure that the entire system is fast and efficient, the modules maintain close interactions via C routines within the Tcl/Tk interpreter.

The object management module, is responsible for reading the data, creating, deleting and editing objects and storing these objects in a hierarchical structure. The hierarchical structure used is very similar to structure used by the DDD, thus the sharing and transfer of data between the two models is fast and efficient. Objects supported by DV include a DIRECTORY type, a POLYGON type, a SPHERE type, and a CYLINDER type. Almost all drawing primitives including points, lines, circles and cones are subclassed under one of the above object types. However, additional object types, such as text, will be incorporated in the future.

The graphics module is composed of a plethora of OpenGL compatible subroutines. To allow for platforms that do not support OpenGL directly, a library called **MESA** translates all OpenGL calls into X11 compatible graphics. Some of the current features incorporated into the graphics include mapping variable data scalars to the alpha channel and stereoscopic viewing. Both of these features will enhance the depth perception associated with a scene to provide a more intuitive view into the data.

The navigator module provides an interface to navigate throughout a world by calling primitive viewing transforms such as rotate, translate and scale. This world/navigator approach allows the user to investigate the data by flying through the world much like a pilot in a flight simulator. Although more complex than using the primitive viewing transforms directly, this approach provides a more versatile viewing environment, especially when large numbers of visual objects are in the world. For example, a scientist using the world/navigator paradigm, can intuitively position the view between objects to investigate an area that might have been obscured had the primitive viewing paradigm been employed.

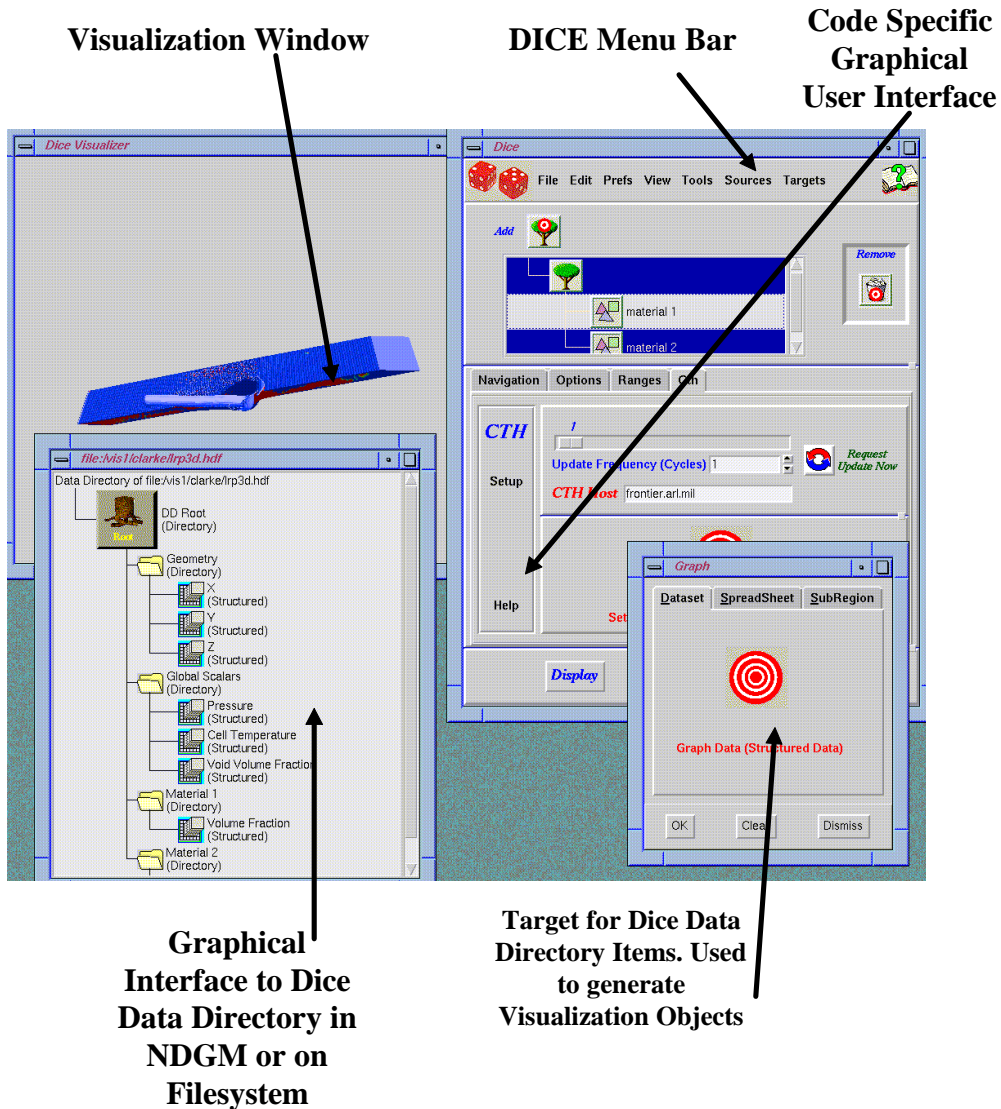
The input module processes user interactions from a variety of input devices, and makes appropriate calls to routines in one of the other DV modules. The input module, like the graphics module, is designed to handle multiple input drivers. Currently, keyboard, GUI, and mouse drivers are supported, in addition to the 3D mouse and head tracker of the ImmersaDesk.

The data filters consist of isosurface generators, and data. The filters will generate unstructured data objects which are written into the DDD, where they can be easily read by DV. They can be applied to a specific timestep of data that has been written to the DDD during the execution of a *DICE* integrated computational code. Thus, while the computational code continues to compute successive timesteps, isosurfaces and cutting planes of the current timestep can be generated and visualized.

Application Interface

The *DICE* user interface consists of several major sections. The main section contains the *DICE* menu bar which is used to control the main functions. From this menu bar the user can open *DICE Data Directories* as sources of data and select action targets. This menu bar is also where the state of the *DICE* navigator is saved and restored.

Below the menu bar is a tree that represents the current contents of the visualizer. *DICE* Visual Objects can be dragged and dropped in order to add objects to the rendered scene.



In the notebook section of the main **DICE** interface window are pages that allow the use to navigate through the data and select drawing preferences. This is also where any code specific GUI is found. A code specific interface usually has a vertical menu bar of its own that allows the user to read and write setup files and to set runtime options.

Main Dice Menu Bar

Menu	Item	Function
File	Restore	Restore Visualization Parameters
	Save	Save Visualization Parameters
	Save Image	Write Current Scene to Image File
	HTTP	Services limited HTTP 1.0 requests
	Connections	Set parameters for remote connections
	Quit	Punt
Edit	Add	Add a new dataset to an existing DDD
	Prune	Delete an existing DDD dataset
Prefs	NDGM	Set NDGM Parameters
	Region	Set the default region for a structured dataset
	Lighting	Lighting parameters
	Stereo	Crystal Eyes stereo and Immersadesk control
	Vis Windows	Color and size of additional visualization windows
View	Local	New local Window
	Remote	New Remote Window
	Tree	New DVO tree
	Viewer	New Viewer
Tools	NDGM	Control and monitor all aspects of NDGM
	Build Dataset	Create a new DDD in NDGM or on Filesystem
Source	File System	DDD on filesystem
	NDGM	DDD in NDGM
Targets	Information	General Information Target
	Spreadsheet	List Data Values for all datasets
	Graph	2D XY plot for structured datasets
	Plane	3D computational plane of structured data
	IsoSurface	3D surface of constant value

The visualization window is part of the initial **DICE** window set. This is an OpenGL window that allows **DICE** to render three dimensional objects. If OpenGL is not present on the platform, **DICE** will accomplish it's visualization via X11 window primitives.

DICE is a drag and drop interface. The concept of data sources and data targets is important to understand. Typically, a **DICE** Data Directory is a source. The DDD can exist in NDGM, on the filesystem, or in both; it is transparent to the interface. Data targets are items like 3D plane generators, 2D graphing interfaces, and isosurface generators.

The hierarchical list of data in the DDD is a metadata object that represents the underlying data. Items from these lists are picked up and dragged into data targets. For example, the 2D graphing target accepts structured datasets, subsets the data, and draws the values in an XY Plot.

Theory of Operation

DICE can be used as a stand alone visualization tool to slice and **DICE** 3D datastes but this is not it's primary function. Typically, **DICE** is used in concert with some other code to look at data as it is produced or to provide a scaleable data format.

A typical **DICE** session consists of the following steps:

- Read a code specific input file to determine data structure size and nature.
- Initialize an NDGM segment.
- Setup a **DICE** Data Directory in the NDGM segment.
- Run a code that periodically writes data to the DDD in NDGM.
- Use the **DICE** drag and drop GUI to look at 2D plots, and 3D planes and isosurfaces.

Zns Interface

The Zns interface is a graphical user interface to the Zonal Navier-Stokes computational fluid dynamics implicit solver. On the Zns page are three major sections : the vertical menu bar, the runtime variable selection and the engine connection section.

Setup of the flow code is accomplished via the menubar. The Setup menu allows the user to read and write input files for the Zns code. These input files are ascii text files that are described in the Zns documentation. The other major functions available through the Setup menu are the ability to initialize NDGM and to produce a batch request file.

The batch request file is necessary when Zns is used on HPC platforms with a batch queue like NQE or GRD. This file is a substitute for interactive requests sent by the GUI. If the file *dicebatch.dat* exists in the current directory, a **DICE** compatible application will assume it to be a file containing update requests. At a frequency specified in the file, the application will update the DDD in NDGM automatically. In this fashion, NDGM and the application can communicate without user intervention.

The Edit menu is used to setup the Zns code. The first menu option allows the user to add zones that are defined in Plot3D compatible grid files. These files are either binary, FORTRAN unformatted, or formatted text. they can be single or multi-grid files.

The second function in the edit menu allows the user to specify calculation parameters. These are values like free stream mach number, Reynolds number, and integration smoothing parameters. The parameters can be specified globally, or on a per zone basis.

The third function in the edit menu is the ability to setup grid overlaps. These are either in a "Pegs" file, in "Chimera files" or specified via the GUI. The GUI is expecting the user to specify exact overlaps in the J, K or L direction. The GUI allows the user to add, modify, and delete these overlaps for each zone. When the user presses the "add" button, the specified overlap is added to the list at the bottom of the GUI. Double clicking on any of the listed overlaps, adjusts all sliders to reflect the parameters. The modify button will change the specified overlap to reflect any changes.

Zones

Zone 2 Boundary Conditions

Boundary Condition Type : Wall (Interpolation)

Dominant Plane Normal Direction

J K L

1 1

J Start 1 J End 1

2 46

K Start 2 K End 46

1 39

L Start 1 L End 39

Boundary Specific Options for Wall (Interpolation)

-1

Time Step Start -1

-1

Time Step End -1

0.0

Spin 0.0

0.0

Add Modify Delete

Type	J Start	J End	K Start	K End	L Start	L End	Condi
Reflective	2	24	1	1	1	89	none
Reflective	2	24	47	47	1	89	none
Polar Axis	1	24	1	47	1	89	none
Outflow	25	25	1	47	1	89	none
Wall (Interpolation)	1	1	2	46	1	39	0 -1 -1
Non Reflective	2	24	2	46	89	89	none

OK Cancel

The final function available in the edit menu is boundary condition definition. This GUI allows the user to specify all available boundary conditions for each zone. If there are any additional information required for a specific condition; like wall temperature for the interpolated wall boundary condition, it is also specified in this GUI. The add, modify, and delete buttons have the same function as in the overlap GUI.

The Run menu allows the user to connect to an executing Zns and to run the Zns code itself. When running the code, the user has the ability to select the number of threads and the frequency at which Plot3D compatible solution files are produced.

To run Zns from **DICE** follow these steps :

- execute “dice zns”
- read in a Zns input file and /or select zone information from the edit-grids menu item
- setup parameters from the edit-calculation parameters menu item
- specify grid overlaps from the edit-overlap menu item
- specify boundary conditions from the edit-boundary condition item
- configure NDGM from the setup
- configure-ndgm item
- select Run Zns from the run-zns menu item
- view planes and isosurfaces by using the drag and drop targets from the targets menu on the main menu bar.

Zns Menu Bar Options

Menu	Item	Function
Setup	Open	Read in an existing Zns input file
	Save	Write a Zns input file
	Create Batch	Create a Dice request file
	NDGM	Configure NDGM for specifically Zns
Edit	Grids	Add additional zone geometry
	Parameters	Problem specific parameters
	Overlaps	Specify Grid Overlaps
	Boundary	Specify Boundary Conditions
	Conditions	
Run	Run Zns	Run flow solver
	Connect	Connect to existing flow solver

CTH Interface

CTH is a computational structural mechanics code from Sandia National Laboratory. The MPI version has been modified to interface with **DICE** and is named dicecth. The **DICE** interface to CTH is used to initialize the NDGM buffer so that CTH can periodically update computed values.

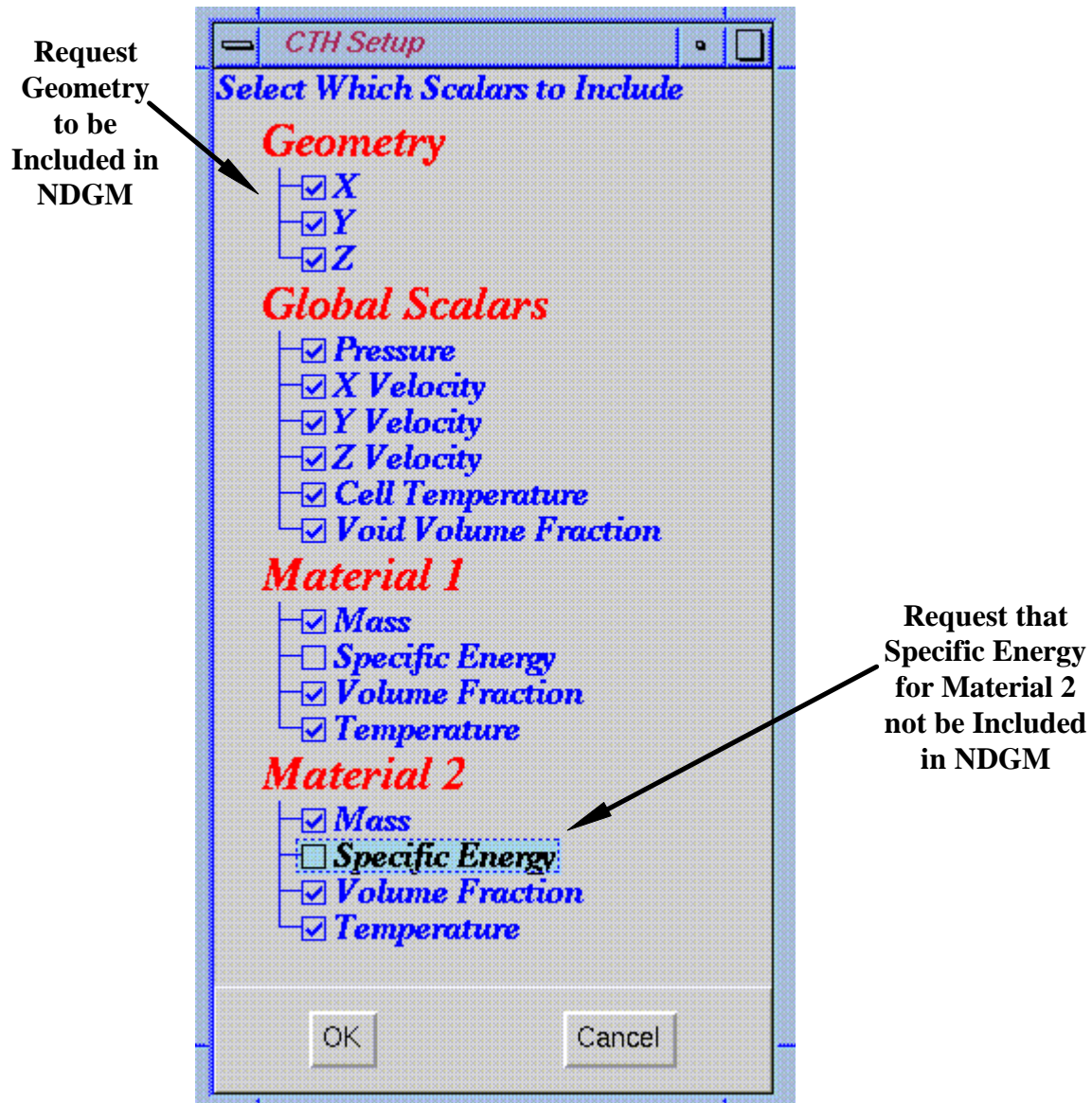
CTHED has also been modified to work with **DICE**. This allows the user to read existing “.plt” files into **DICE** in order to post process results. The modified CTHED is called dicecthst and will only output certain quantities.

To use **DICE** with CTH :

- Select an “ogcth” file which was output from cthgen
- Choose which variable you wish to store in the DDD in NDGM

- Run dicecth and interactively request an update
or
- Run dicecthrst with a batch update request file

Sandia has developed an input GUI for CTH written in Tcl/Tk. This interface has been incorporated into the [DICE](#) CTH page. It has not yet been fully integrated into [DICE](#). Until it's fully integrated, use the cthgen output file to setup NDGM.



[DICE](#) reads an input file in order to determine the number of materials and the domain dimensions. The user is presented with a tree widget with which to select the variable that are to be stored into the DDD in NDGM. [DICE](#) then creates an NDGM buffer of appropriate size and configured the DDD.

If running dicecthrst or running dicecth in a batch environment, the user needs to create a batch update file. This tells the application what variable in the DDD to update and how often this update is to occur. The user uses a tree widget to select which variable to update and sets the slider to the appropriate update frequency. The filename “dicebatch.dat” is generally chosen as the name of the batch update file. This is because so that not all *DICE* compatible applications need modify the input argument list.

At every requested cycle, each MPI CTH process updates the NDGM buffer with its portion of the data. In this way the entire dataset is reconstructed from the various portions. Once updated, the dataset can be interrogated with 2D plots and 3D surfaces.

CTH Menu Bar Items

Menu	Item	Function
Setup	From CTH Input	CTH input GUI from Sandia
	From CTHGEN Output	Setup NDGM from CTHGEN output file
	Batch Update File	Create DICE request file
	NDGM	Configure NDGM
	Connect to CTH	Connect to running CTH

Other

Through experience gained while incorporating into current applications, a *DICE* Runtime Library is being developed. This reduces the effort to make other applications *DICE* compatible. Currently this library deals with structured datasets but support for unstructured codes will be coming soon.

The *DICE* visualization tools are primarily designed for fast, runtime interrogation of large datasets. For more traditional post processing of data, a more traditional scientific visualization tool like EnSight from CEI can be used. The utility “dice_to_ensight” can be used to convert data in a DDD to EnSight6 format.

For more information and API references, please visit the DICE web page.

For Additional Information :

<http://frontier.arl.mil/clarke/dice.html>

DICE

Jerry A. Clarke

clarke@arl.mil

Raytheon E-Systems

U.S. Army Research Laboratory

Major Shared Resource Center (PET)

Jennifer J. Hare

jen@arl.mil

U.S. Army Research Laboratory

Scientific Visualization Team

Charles E. Schmitt

erics@arl.mil

Network Computing Services Inc.

U.S. Army Research Laboratory

Army High Performance Computing Research Center

Zns

James P. Collins

pcollins@arl.mil

U.S. Army Research Laboratory

Harris L. Edge

edge@arl.mil

U.S. Army Research Laboratory

Jerry A. Clarke

clarke@arl.mil

Raytheon E-Systems

U.S. Army Research Laboratory

Major Shared Resource Center (PET)

CTH

Eugene Hertel

eshert@sandia.gov

Sandia National Laboratories

Computational Physics Research, Development
and Applications Dept.